RBE/CS549: Homework 0 - Alohomora

Shrishailya Chavan
WPI Robotics Engineering
schavan@wpi.edu
Using 1 late day

***Abstract*—Here I have developed an algorithm of pb(probability of boundary) boundary detection algorithm, which finds boundaries by examining brightness, color, and texture information across multiple scales. The output is a per-pixel probability of boundary. Furthermore, I have also trained and tested model using CIFAR-10 dataset on various models such as Simple Neural Network, Modified Neural Network, ResNet, ResNext and DenseNet.**

***Index Terms*—Probability-based edge detection, Convolutional Neural Networks, ResNet, ResNEXT, DenseNet, Pytorch.**

I. PHASE 1: SHAKE MY BOUNDARY

This section is the implementation of the pb boundary detection algorithm introduced. It is somewhat different from the classical CV techniques that are universally used all over like in Sobel and Canny Filters it uses the texture and color information present in the image in addition to the intensity discontinuities as well. This is done in 4 steps in the sections to follow:

1) Filter Banks
2) Texture, Brightness and Color Maps T, B, C
3) Texture, Brightness and Color Gradients Tg, Bg, Cg
4) Pb-lite output combined with baselines

We will be going through all the above steps mentioned step by step.

A. *Filter Banks*

Here, the first step of the given Pb lite boundary detection is to filter out the images using the set of filter banks. Following we have three different sets of filter banks for this purpose. The use of these filters on images with these banks will help us to build the low level features which represent texture.

*1) **Oriented DoG Filter Bank**:* Oriented DoG Filter Bank are created by convolving a simple Sobel Filter and a Gaussian kernel and further rotating results. Here I have total 2 scales and 16 different orientations. The Filter bank has scale of 3 and 4 with kernel size 81. Fig. 1 shows these filters.

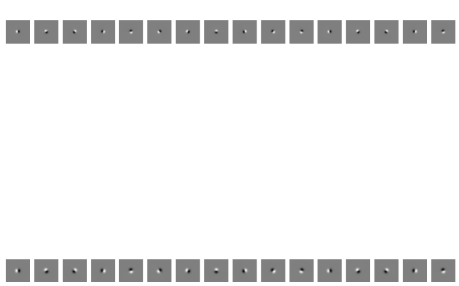


Fig. 1. DoG filters

*2) **Leung-Malik Filter Bank**:* The Leung-Malik filters are a set of multi scale, multi oreintation filter bank with 48 filters. It consists of first and second order derivatives of Gaussians at 6 orientations and 3 scales making a total of 36; 8 Laplacian of Gaussian (LoG) filters; and 4 Gaussians. In LM Small (LMS), the filters occur at basic sigma scales $(1, \sqrt{2}, 2, 2\sqrt{2}\ )$. The first and second derivative filters occur at the first three scales with an elongation factor of 3. The Gaussians occur at the four basic scales while the 8 LOG filters occur at sigma and 3sigma. For LM Large (LML), the filters occur at the basic sigma scales $(\sqrt{2}, 2, 2\sqrt{2}, 4)$. Fig. 2 represent these filters.

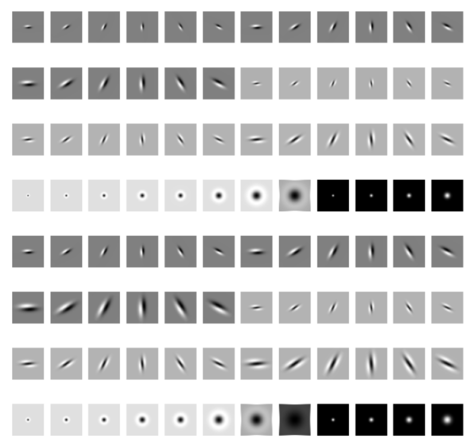


Fig. 2. LM filters

*3) **Gabor Filter Bank**:* Gabor filters are designed on the filters in the human visual system. A Gabor filter is a gaussian kernel function modulated by a sinusoidal plane wave. The Gabot filter has scale of 8, 16 and 24, it also has 8 orientations with 2, 4, 6 frequencies and kernel size of 49. Fig. 3 shows these filters

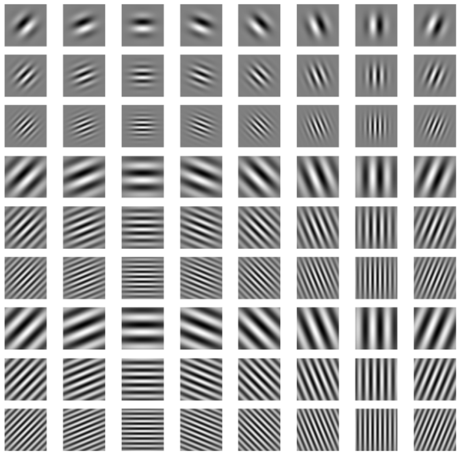


Fig. 3. Gabor filters

B. *Texton, Brightness, Color maps- T, B, C*

*1) **Texton map**:* Filtering an input image with each element of the filter bank results in a vector of filter responses centered on each pixel. A distribution of these N-dimensional filter responses could be thought of as encoding texture properties. Here we have simplified this representation by replacing each Ndimensional vector with a discrete Texton ID. We do this by clustering the filter responses at all pixels in the image into K Textons using K-means clustering. Each pixel is then represented by a one dimensional, discrete cluster ID instead of a vector of high-dimensional, real-valued filter responses. This is then represented in the form of a single channel image with values in the range of [1,2,3,...,K].K = 64. After that it was observed that the filtered output of the images were low intensity values that caused poor clustering. So, after filtering the output image was normalized in the range of 0-255 which improved the clustering.

*2) **Brightness Map**:* The concept of the brightness map is as simple as capturing the brightness changes in the image. Here, again we cluster the brightness values(gray scale equivalent of the color image) using kmeans clustering into a chosen number of clusters (K=16). We call the clustered output as the brightness map B.

*3) **Color Map**:* The concept of the color map is to capture the color changes or chrominance content in the image. Here, again we cluster the RGB color values using kmeans clustering into a chosen number of clusters (K=16). We call the clustered output as the color map C. Figures 4 through 13 show these texton, brightness and color maps for the ten different test images.

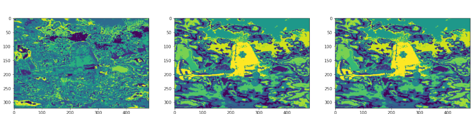


Fig. 4. T, B, C maps for image 1

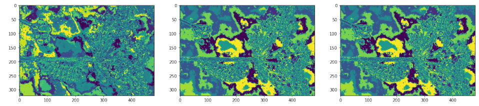


Fig. 5. T, B, C maps for image 2

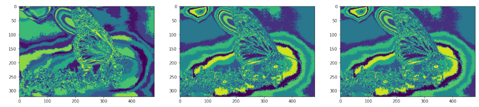


Fig. 6. T, B, C maps for image 3

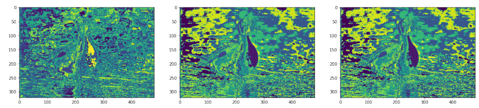


Fig. 7. T, B, C maps for image 4

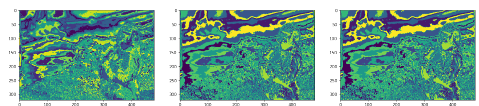


Fig. 8. T, B, C maps for image 5

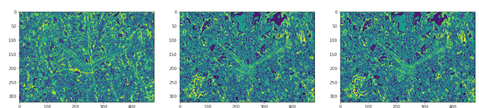


Fig. 9. T, B, C maps for image 6

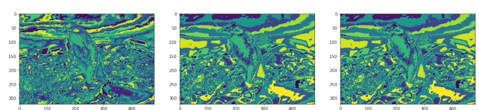


Fig. 10. T, B, C maps for image 7

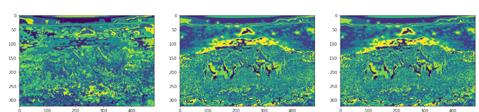


Fig. 11. T, B, C maps for image 8

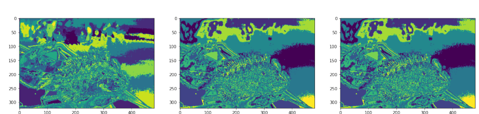


Fig. 12. T, B, C maps for image 9

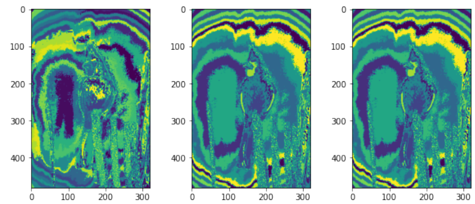


Fig. 13. T, B, C maps for image 10

C. Texture, Brightness and Color Gradients- Tg, Bg, Cg

Texture, Brightness and Color gradients encode how much the texture, brightness and color distributions are changing at a pixel. To obtain Tg, Bg, and Cg, we need to compute differences of values across different shapes and sizes. This can be achieved very efficiently by the use of Half-disk masks.

*1) **Half Disk Masks**:* The half-disc masks are simply (pairs of) binary images of half-discs. These allow us to compute the chi-square distances using a filtering operation, which is much faster than looping over each pixel neighborhood and aggregating counts for histograms. Forming these masks is quite trivial. The set of masks used for this work with 8 orientations and 3 scales is shown in Fig. 14

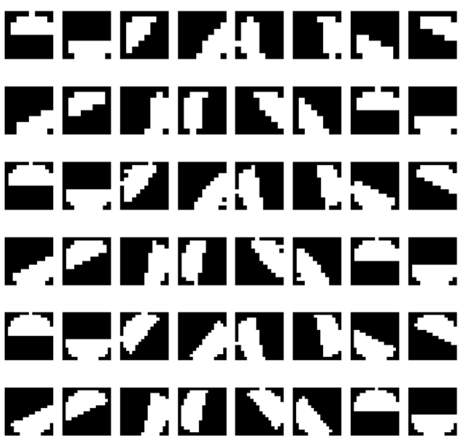


Fig. 14. Half-disc masks

We compute Tg, Bg, Cg by comparing the distributions in left/right half-disc pairs (opposing directions of filters at same scale) centered at a pixel. If the distributions are the similar, the gradient should be small. If the distributions are dissimilar, the gradient should be large. Because our half-discs span multiple scales and orientations, we will end up with a series of local gradient measurements encoding how quickly the texture or brightness distributions are changing at different scales and angles. We will compare texton, brightness and color distributions with the chi-square measure. The chi-square distance is a frequently used metric for comparing two histograms.

Figures 15 through 24 show the Tg, Bg, Cg gradients for the ten test images.

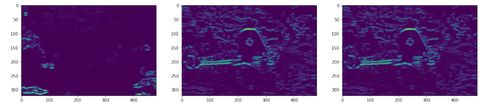


Fig. 15. Tg, Bg, Cg gradients for image 1

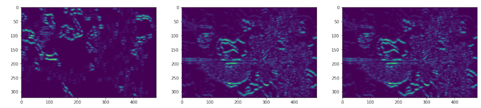


Fig. 16. Tg, Bg, Cg gradients for image 2

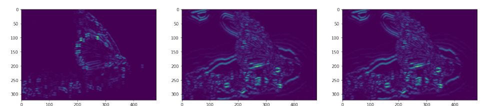


Fig. 17. Tg, Bg, Cg gradients for image 3

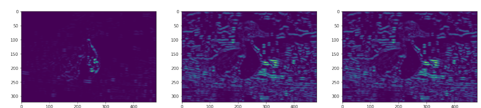


Fig. 18. Tg, Bg, Cg gradients for image 4

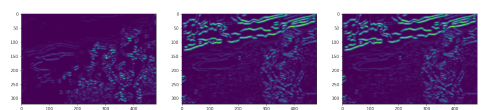


Fig. 19. Tg, Bg, Cg gradients for image 5

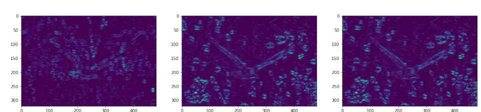


Fig. 20. Tg, Bg, Cg gradients for image 6

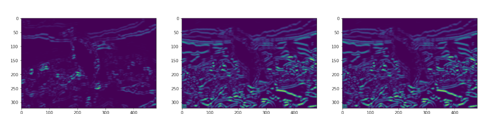


Fig. 21. Tg, Bg, Cg gradients for image 7

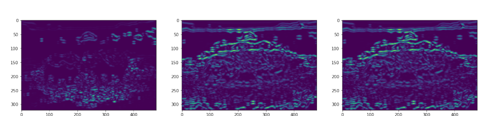


Fig. 22. Tg, Bg, Cg gradients for image 8

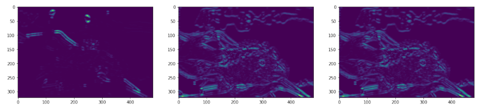


Fig. 23. Tg, Bg, Cg gradients for image 9

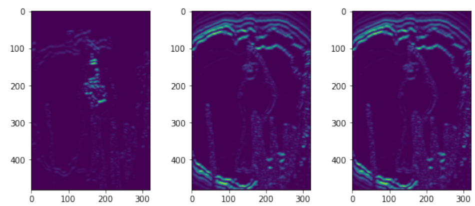


Fig. 24. Tg, Bg, Cg gradients for image 10

D. *Pb-lite output combined with baselines*

Figures 25 through 34 show the final pb-lite output combined with the canny and sobel baselines (both baselines have been given equal weightage), alongside the original canny and sobel results for comparison. It is observed that pb-lite edges are lacking of most of the nosie that canny and sobel contain. The reason is that it is good at suppressing the falsen positives which show up the noise in sobel and canny. The final output is better and can can be improved by changing the filters and looking for the filter that can perform better than the existing ones.

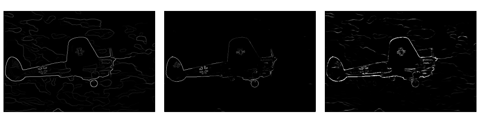


Fig. 25. Canny, Sobel, Pb-lite for image 1

Fig. 26. Canny, Sobel, Pb-lite for image 2

Fig. 27. Canny, Sobel, Pb-lite for image 3

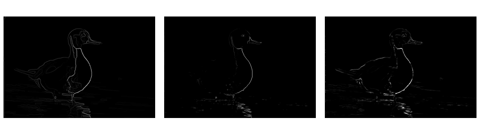


Fig. 28. Canny, Sobel, Pb-lite for image 4

Fig. 29. Canny, Sobel, Pb-lite for image 5

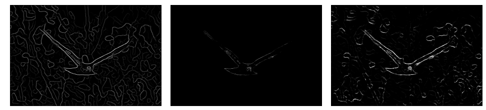


Fig. 30. Canny, Sobel, Pb-lite for image 6

Fig. 31. Canny, Sobel, Pb-lite for image 7

Fig. 32. Canny, Sobel, Pb-lite for image 8

Fig. 33. Canny, Sobel, Pb-lite for image 9

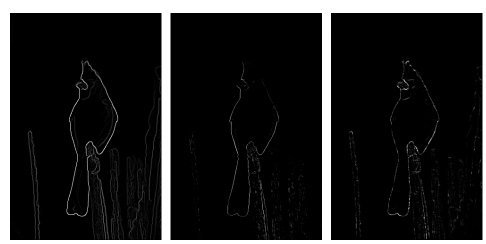


Fig. 34. Canny, Sobel, Pb-lite for image 10

REFERENCES

[1] http://www.robots.ox.ac.uk/ vgg/research/texclass/filters.html
[2] https://stackoverflow.com/questions/55013954/how-to-apply-a-gabor-filter-to-an-image-with-hexagonal-sampling
[3] https://www.delftstack.com/howto/python/gaussian-kernel-python/

II. PHASE 2: DEEP DIVE ON DEEP LEARNING

A. *My first neural network*

I am quite good at implementing neural networks and have already studied Deep Learning in my first semester. I tried to implement the simple neural network. There were a few bugs in the starter code, but after debugging them I was able to implement the most basic Neural Network following the instructions in the assignment. The architecture for my network is shown in the figure. I used the Adam Optimizer with a learning rate of 0.001, and let the network train for 30 Epochs with a batch size of 25. I also applied MaxPooling to all the layers. Furthermore, I used CrossEntropy loss function. After each epoch I calculate the mean loss and accuracy and plotted them accordingly. The number of parameters in the network are 25218. The test set was able to predict the image with an accuracy of 14.04%. This is understandable since I just simply wanted to train a neural network without any complications. The architecture is shown in figure 35. The confusion matrix for the training set and testing set is shown in figure 36, 37. Additionally, I have also plotted the Test accuracy against total number of epochs.

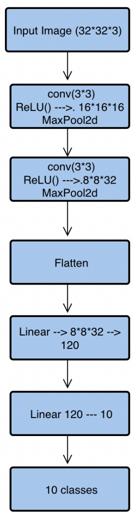
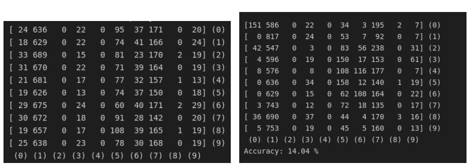


Fig. 35. My Simple CNN



```
[ 24 636   0  22   0  95  37 171   0  20] (0)
[ 18 629   0  22   0  74  41 166   0  24] (1)
[ 33 689   0  15   0  81  23 170   2  19] (2)
[ 31 670   0  22   0  71  39 164   0  19] (3)
[ 21 681   0  17   0  77  32 157   1  13] (4)
[ 19 626   0  13   0  74  37 150   0  18] (5)
[ 29 675   0  24   0  60  40 171   2  29] (6)
[ 30 672   0  18   0  91  28 142   0  20] (7)
[ 19 657   0  17   0 108  39 165   1  19] (8)
[ 25 638   0  23   0  78  30 168   0  19] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

```
[151 586   0  22   0  34   3 195   2   7] (0)
[  0 817   0  24   0  53   7  92   0   7] (1)
[ 42 547   0   3   0  83  56 238   0  31] (2)
[  4 596   0  19   0 150  17 153   0  61] (3)
[  8 576   0   8   0 108 116 177   0   7] (4)
[  0 636   0  34   0 158  12 140   1  19] (5)
[  0 629   0  15   0  62 108 164   0  22] (6)
[  3 743   0  12   0  72  18 135   0  17] (7)
[ 36 690   0  37   0  44   4 170   3  16] (8)
[  5 753   0  19   0  45   5 160   0  13] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 14.04 %
```

Fig. 36. (a) Train Confusion Matrix (b) Test Confusion Matrix for SimpleNN

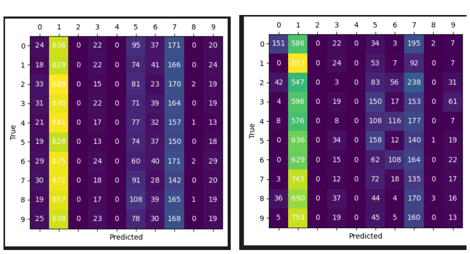


Fig. 37. (a) Train Confusion Matrix (b) Test Confusion Matrix for SimpleNN

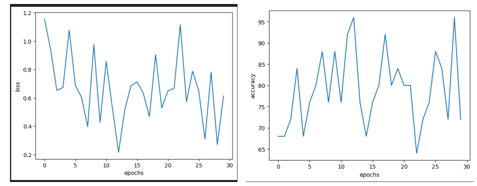


Fig. 38. Train - Loss, Acc vs epochs for SimpleNN

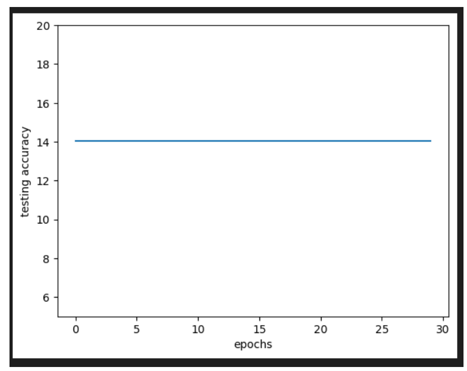


Fig. 39. Test- Acc vs epoch for SimpleNN

B. *Improved Network*

To improve my simple Neural Network I added some features and applied Standardization to try and improve accuracy. I also applied Batch Normalization to the layers here to increase the accuracy. I kept epochs at 30. Batch size was also 25 for this network too. The results are shown in figures. The number of parameters are 147794. Here, I used the Adam optimizer with learning rate of 0.001. After improving the simple neural network by applying various methods I was able to improve my accuracy by some extent(I was expecting to increase it a bit more but it didn't), I think I need to make some more changes by increasing number of epochs and adding more layer to my network which will result in increasing the accuracy of my network. The Testing accuracy that I got was 20.59%. I have plotted the mean Training and loss against the total number of epochs, calculated the Testing and Training Confusion Matrix with their respective tables. The Testing accuracy was also plotted against the total number of epochs.

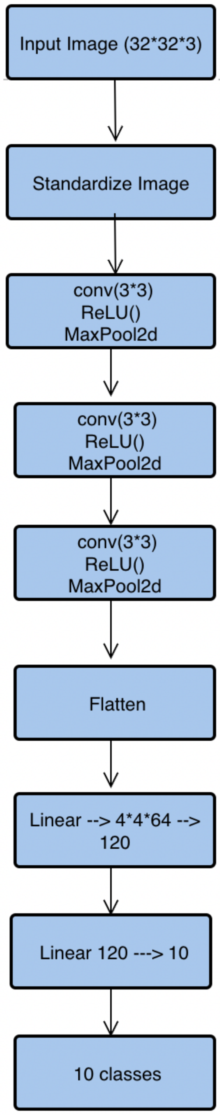
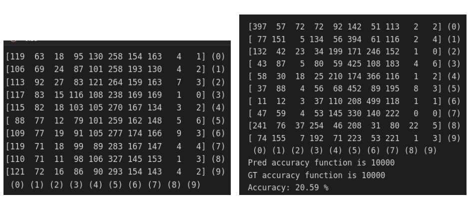


Fig. 40. Architecture for my Improved NN



```
[119  63  18  95 130 258 154 163   4   1] (0)
[106  69  24  87 101 258 193 130   4   2] (1)
[113  92  27  83 121 264 159 163   7   3] (2)
[117  83  15 116 108 238 169 169   1   0] (3)
[115  82  18 103 105 270 167 134   3   2] (4)
[ 88  77  12  79 101 259 162 148   5   6] (5)
[109  77  19  91 105 277 174 166   9   3] (6)
[119  71  18  99  89 283 167 147   4   4] (7)
[110  71  11  98 106 327 145 153   1   3] (8)
[121  72  16  86  90 293 154 143   4   2] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

```
[397  57  72  72  92 142  51 113   2   2] (0)
[ 77 151   5 134  56 394  61 116   2   4] (1)
[132  42  23  34 199 171 246 152   1   0] (2)
[ 43  87   5  80  59 425 108 183   4   6] (3)
[ 58  30  18  25 210 174 366 116   1   2] (4)
[ 37  88   4  56  68 452  89 195   8   3] (5)
[ 11  12   3  37 110 208 499 118   1   1] (6)
[ 47  59   4  53 145 330 140 222   0   0] (7)
[241  76  37 254  46 208  31  80  22   5] (8)
[ 74 155   7 192  71 223  53 221   1   3] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Pred accuracy function is 10000
GT accuracy function is 10000
Accuracy: 20.59 %
```

Fig. 41. (a) Train Confusion Matrix (b) Test Confusion Matrix for Improved NN

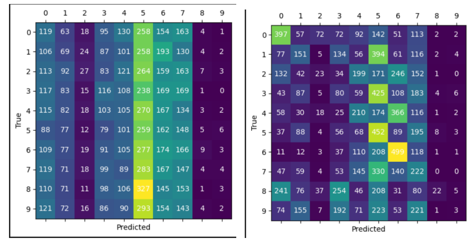


Fig. 42. (a) Train Confusion Matrix (b) Test Confusion Matrix for Improved NN

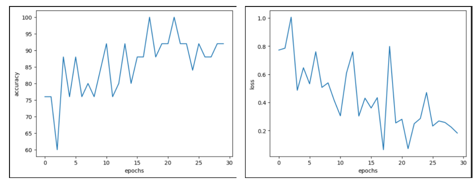


Fig. 43. Train - Loss, Acc vs epochs for Improved NN

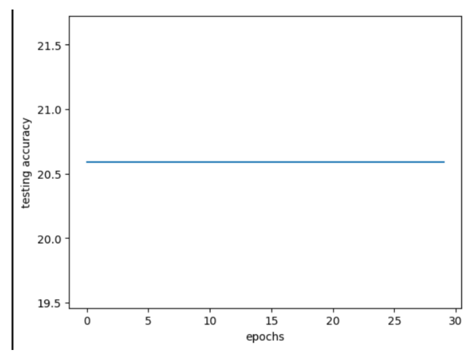


Fig. 44. Test- Acc vs epoch for Improved NN

C. *ResNET*

Further, I implemented the ResNET to increase my accuracy compared to the previous two implemented simple networks. The network implemented has 3, 3, 6 and 3 layers respectively, further we developed a Residual Block to implement our ResNET Network. The whole idea behind the ResNET is to skip the layers which the normal NN usually do, which requires a lot of computation. The architecture is shown below in figure 45. The general NN try to approximate the ideal solution by doing backward propogation and hypertune the weights. But as the gradient starts to vanish, you start to oscillate and never converge to a solution. Thus, instead of calculating approximate value to the idea solution, ResNET takes the ideal solution and tries to approximate the residual by skipping some of the layers. Please read up the papers for the exact details. I trained ResNET with Batch Size of 25, did data standardization same as improved network, number of epochs = 30, also used weight decay as 0.001 as a part of Adam optimizer. The number of parameters of ResNET are 21298314. The accuracy for testing was around 14.98% which is pretty low. My training accuracy is more than testing accuracy which might be because there might be meaningful differences between the kind of data I trained the model on and the testing data that I am providing for evaluation which I need to figure out. Furthermore, I think the reason for low accuracy might as well be that I am training my model on low epochs but my model was consistent and works properly as I have followed the official documentation. For this model I have plotted the Training, Testing and loss with respect to total epochs. Furthermore, I have also calculated the Testing and Training Confusion Matrix as well.

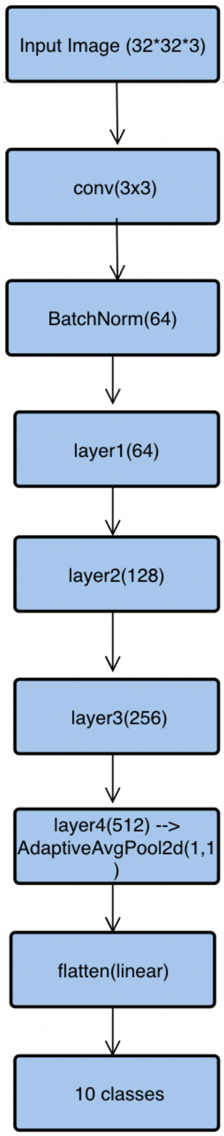
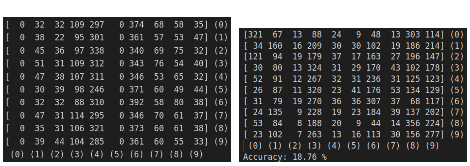


Fig. 45. Architecture of ResNET



```
[  0  32  32 109 297   0 374  68  58  35] (0)
[  0  38  22  95 301   0 361  57  53  47] (1)
[  0  45  36  97 338   0 340  69  75  32] (2)
[  0  51  31 109 312   0 343  76  54  40] (3)
[  0  47  38 107 311   0 346  53  65  32] (4)
[  0  30  39  98 246   0 371  60  49  44] (5)
[  0  32  32  88 310   0 392  58  80  38] (6)
[  0  47  31 114 295   0 346  70  61  37] (7)
[  0  35  31 106 321   0 373  60  61  38] (8)
[  0  39  44 104 285   0 361  60  55  33] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

```
[321  67  13  88  24   9  48  13 303 114] (0)
[ 34 160  16 209  30  30 102  19 186 214] (1)
[121  94  19 179  37  17 163  27 196 147] (2)
[ 30  80  13 324  31  29 170  43 102 178] (3)
[ 52  91  12 267  32  31 236  31 125 123] (4)
[ 26  87  11 320  23  41 176  53 134 129] (5)
[ 31  79  19 270  36  36 307  37  68 117] (6)
[ 24 135   9 228  19  23 184  39 137 202] (7)
[ 53  84   8 188  20   9  44  14 356 224] (8)
[ 23 102   7 263  13  16 113  30 156 277] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 18.76 %
```

Fig. 46. (a) Train Confusion Matrix (b) Test Confusion Matrix for ResNET

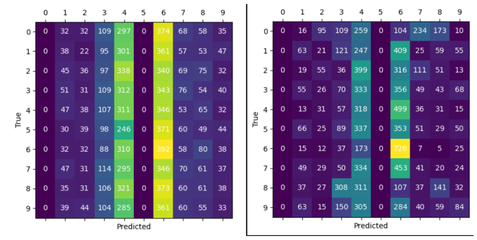


Fig. 47. (a) Train Confusion Matrix (b) Test Confusion Matrix for ResNET

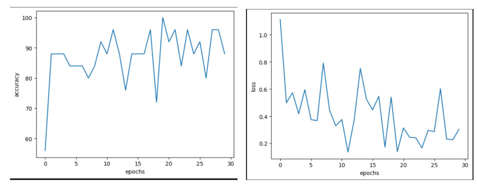


Fig. 48. Train - Loss, Acc vs epochs for ResNET

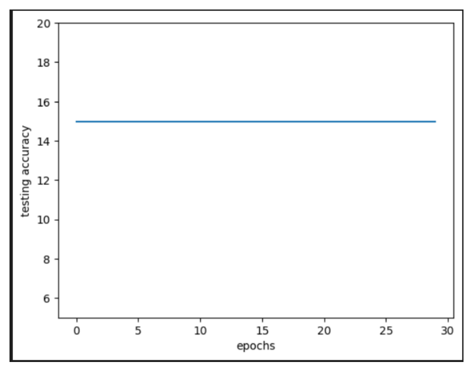
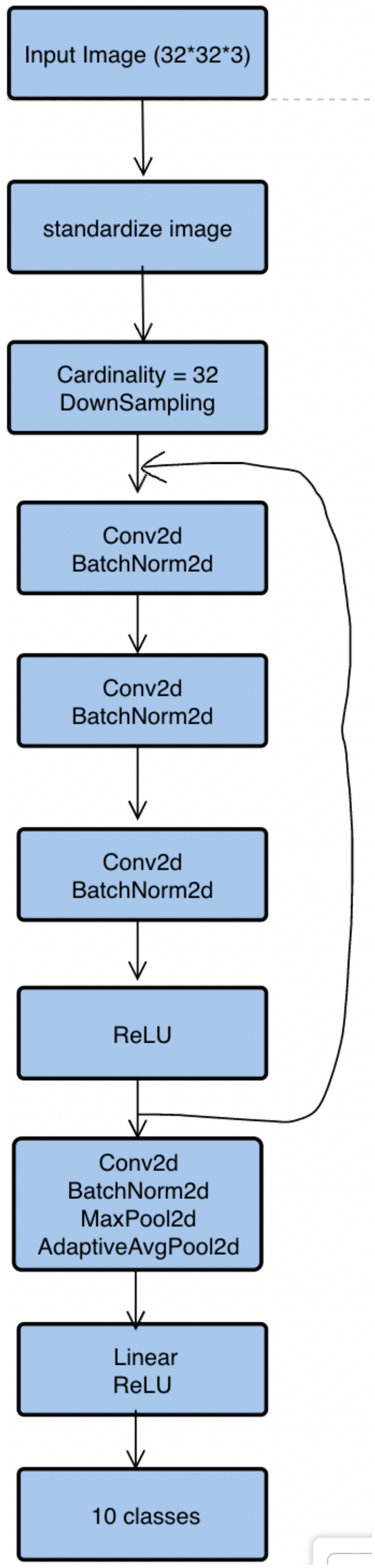


Fig. 49. Test- Acc vs epoch for ResNET

D. *ResNEXT*

ResNEXT Neural Network is a slight improvement on the ResNET model. According to the paper they add one more layer to the ResNET model for an improvmenet of 0.2-0.3% accuracy. As required I got the accuracy for ResNext model greater than that of ResNET model. The Testing accuracy I got for my ResNEXT model is 18.76% I implemented the ResNEXT 50 model here with cardinality equals to 32 which is mentioned in the offical paper. The total number of parameters that I got in this are 25028904. I used the Adam optimizer with weight decay of 0.001 for improving the results. The total number of epochs for this are 30 with batch size of 25. For this model I have plotted the Training, Testing and loss with respect to total epochs. Furthermore, I have also calculated the Testing and Training Confusion Matrix as well.



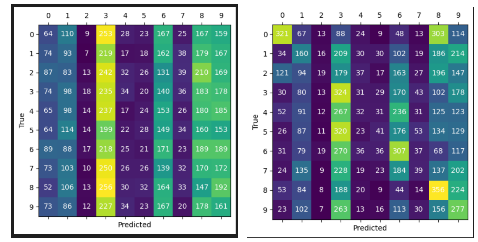
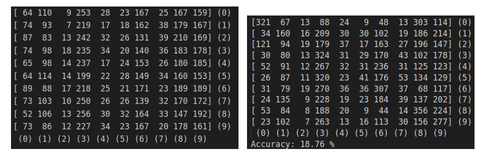


Fig. 51. (a) Train Confusion Matrix (b) Test Confusion Matrix for ResNEXT



```
[ 64 110   9 253  28  23 167  25 167 159] (0)
[ 74  93   7 219  17  18 162  38 179 167] (1)
[ 87  83  13 242  32  26 131  39 210 169] (2)
[ 74  98  18 235  34  20 140  36 183 178] (3)
[ 65  98  14 237  17  24 153  26 180 185] (4)
[ 64 114  14 199  22  28 149  34 160 153] (5)
[ 89  88  17 218  25  21 171  23 189 189] (6)
[ 73 103  10 250  26  26 139  32 170 172] (7)
[ 52 106  13 256  30  32 164  33 147 192] (8)
[ 73  86  12 227  34  23 167  20 178 161] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

```
[321  67  13  88  24   9  48  13 303 114] (0)
[ 34 160  16 209  30  30 102  19 186 214] (1)
[121  94  19 179  37  17 163  27 196 147] (2)
[ 30  80  13 324  31  29 170  43 102 178] (3)
[ 52  91  12 267  32  31 236  31 125 123] (4)
[ 26  87  11 320  23  41 176  53 134 129] (5)
[ 31  79  19 270  36  36 307  37  68 117] (6)
[ 24 135   9 228  19  23 184  39 137 202] (7)
[ 53  84   8 188  20   9  44  14 356 224] (8)
[ 23 102   7 263  13  16 113  30 156 277] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 18.76 %
```

Fig. 52. (a) Train Confusion Matrix (b) Test Confusion Matrix for ResNEXT

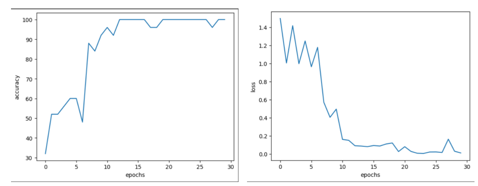


Fig. 53. Train - Loss, Acc vs epochs for ResNEXT

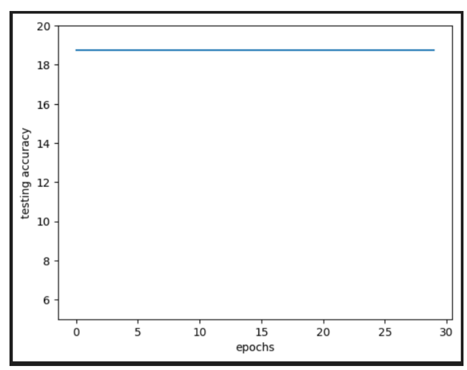


Fig. 54. Test- Acc vs epoch for ResNEXT

E. DenseNet

DenseNet is an interesting NeuralNetwork. The idea that if there are L layers in the network, the Lth layer gets $L^*(L+1)/2$ inputs to it. The DenseNet also makes use of Denseblocks, denselayers and transition layers which are used to transform the information into different sizes. For this model I got an Testing accuracy around 11.82% which was because I implemented the most basic DenseNet model. The other reason might be the less number of epochs on which I am training my model. I trained it on 30 epochs with batch size of 25. I used Adam optimizer with weight decay of 0.001 to increase the accuracy. The total number of parameters in this DenseNet are 1071946. For this model I have plotted the Training, Testing and loss with respect to total epochs. Furthermore, I have also calculated the Testing and Training Confusion Matrix as well. Further, by implementing more layers and making some amends in Network we can increase the accuracy.

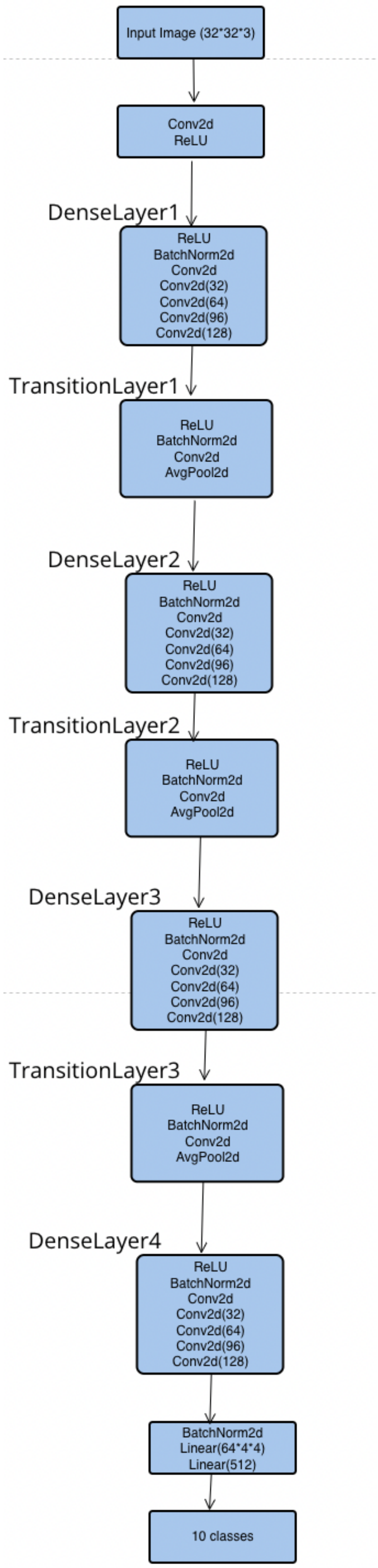


Fig. 55. Architecture for DenseNet

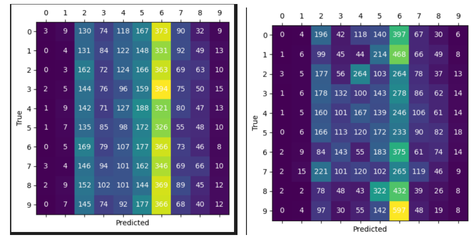
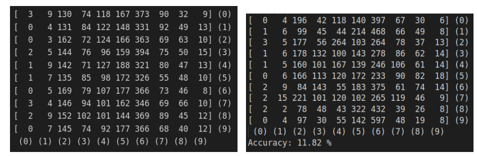


Fig. 56. (a) Train Confusion Matrix (b) Test Confusion Matrix for DenseNet



```
[  3   9 130  74 118 167 373  90  32   9] (0)
[  0   4 131  84 122 148 331  92  49  13] (1)
[  0   3 162  72 124 166 363  69  63  10] (2)
[  2   5 144  76  96 159 394  75  50  15] (3)
[  1   9 142  71 127 188 321  80  47  13] (4)
[  1   7 135  85  98 172 326  55  48  10] (5)
[  0   5 169  79 107 177 366  73  46   8] (6)
[  3   4 146  94 101 162 346  69  66  10] (7)
[  2   9 152 102 101 144 369  89  45  12] (8)
[  0   7 145  74  92 177 366  68  40  12] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

```
[  0   4 196  42 118 140 397  67  30   6] (0)
[  1   6  99  45  44 214 468  66  49   8] (1)
[  3   5 177  56 264 103 264  78  37  13] (2)
[  1   6 178 132 100 143 278  86  62  14] (3)
[  1   5 160 101 167 139 246 106  61  14] (4)
[  0   6 166 113 120 172 233  90  82  18] (5)
[  2   9  84 143  55 183 375  61  74  14] (6)
[  2  15 221 101 120 102 265 119  46   9] (7)
[  2   2  78  48  43 322 432  39  26   8] (8)
[  0   4  97  30  55 142 597  48  19   8] (9)
 (0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
Accuracy: 11.82 %
```

Fig. 57. (a) Train Confusion Matrix (b) Test Confusion Matrix for DenseNet

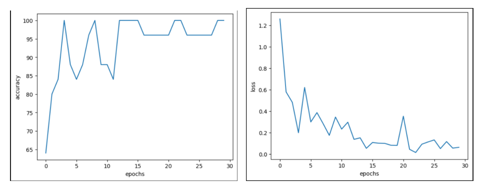


Fig. 58. Train - Loss, Acc vs epochs for DenseNet

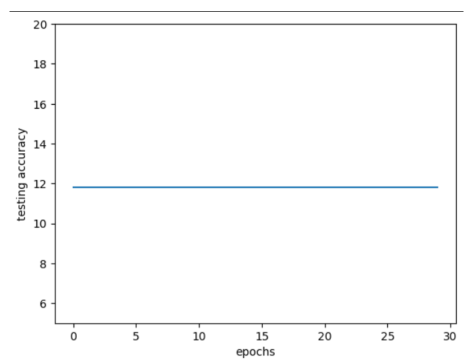


Fig. 59. Test- Acc vs epoch for DenseNet

F. Analysis

From all the networks that I developed and tested, the number of parameters, all the required graphs, accuracy for 30 epochs with batch size of 25 are shown here. I got more accuracy for my Improved Neural Network. Among ResNet, ResNEXT and DenseNet I got more accuracy for ResNEXT which is improved model of ResNet. The Training accuracy for all models was more than that of Testing accuracy, the reason might be that there are some meaningful differences between the kind of data I trained the model on and the testing data which I am providing for evaluation. Maybe I need to tune some hyperparameters to fix it. I will further perform the Data Augmentation and various other tricks so that the Testing accuracy increases. For the above all models I have calculated the Testing and Training Confusion Matrix as well. Further, by implementing more layers and making some amends in Network we can increase the accuracy. I think models are good enough to get a good accuracy, I need to work on other parameters to increase the accuracy of my model and train it on high computations machines to run them smoothly.

REFERENCES

[1] https://medium.com/jovianml/using-resnet-for-image-classification-4b3c42f2a27e

[2] https://arxiv.org/pdf/2110.04632v1.pdf

[3] https://medium.com/jovianml/using-resnet-for-image-classification-4b3c42f2a27e